

# **Application Note**

**XYZ Company Microcontroller Products**

## **Designing for Low Power with XYZ Microcontrollers**

---

(Date)

Author: CalaRos Bay Systems (<http://www.calarosbay.com>)

## Abstract

The XYZ™ microcontrollers are packed with features that can help reducing the power consumption of systems built around them.

Reducing the power consumption of a system is very application dependent; this document presents different venues to explore, and provides recommendations with regards to both hardware and software designs that feature an XYZ™ microcontroller.

For instance, using memory mapping instead of GPIO may provide significant power consumption savings, depending upon the profile of the application.

On another hand, while reducing the clock frequency is often a chosen option, this document sheds light on the fact that it may not be appropriate in some cases.

Other more general power savings techniques are also described, such as reducing PCB trace length and partitioning busses, pulsing LEDs and displays, and using Look-Up-Tables.

When applicable, this document provides power savings estimates, and examples of XYZ™ microcontroller register settings that can help reducing the power consumption of the system.

## Overview

Low power consumption comes in naturally as a key goal for mobile battery operated systems.

It is also very important for resident systems that draw their energy from limited power supplies such as solar panels, or when in space and temperature constraint environments.

Further down the line, a 2000 survey from the Energy Conservation Center of Japan (ECCJ) points out that digital appliances (e.g. digital videotape decks, videodisk recorders, satellite tuners, digital home terminals for cable broadcasting, and hard disk recorders) tend to have a higher power consumption when in sleep mode than traditional analog ones; thus the trends of power consumption reduction in technical development for digital appliances should be watched carefully.

The XYZ™ microcontrollers (XYZ123000, XYZ12300x, XYZ12400x) are geared towards this approach, as they combine the low power architecture of the ARM® core with a state of the art silicon technology, allowing for low power consumption when in operation.

They also offer specific features that are detailed in this document, and that help further reduce the power consumption of any system over long periods of time.

This document doesn't provide a definite recipe for low power consumption, but rather suggests different paths to explore and balance, depending upon the specifics of the system.

Hunting down micro-watts is a state of mind; the XYZ™ microcontrollers have it already.

## 1- Hardware Design tips

### 1.1- Take advantage of Memory Mapping for external I/O access:

Although using GPIO is an option to access peripheral devices, the XYZ™ microcontrollers offer a much better one: *memory mapping of the I/O through the External Memory Controller*.

Memory mapping allows defining your peripheral as a memory location and access it in a very power efficient manner (see figure 1 below).

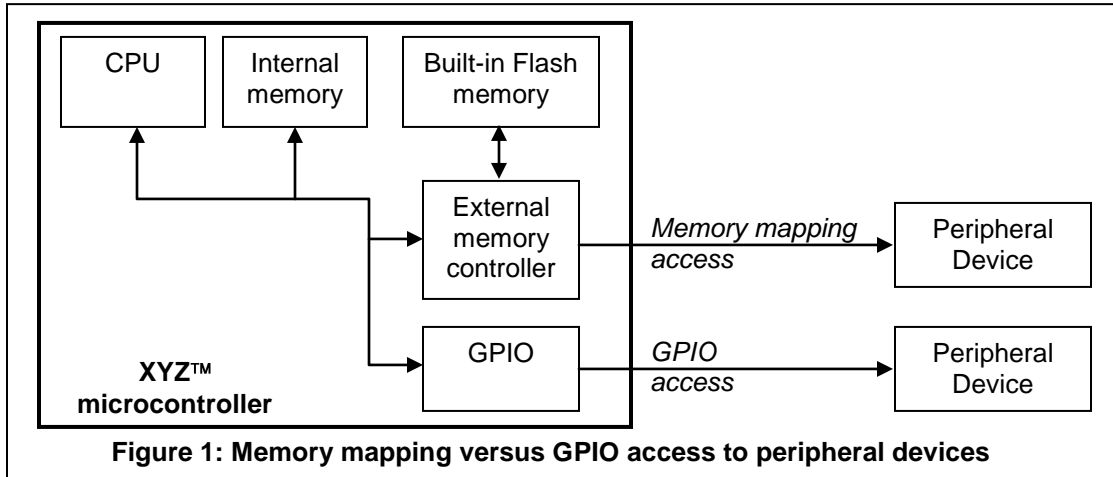


Figure 1: Memory mapping versus GPIO access to peripheral devices

Experiments done at XYZ show that the memory mapping method allows an 88% performance improvement over a GPIO access method.

Faster I/O operations allow completing tasks sooner, and the microcontroller can then be put in Standby mode more often, saving power (see paragraph 1.3 below).

### 1.2- Memory mapping for external I/O access - How to:

The table below shows an example of configuration for a 16-bits bus width external peripheral connected to Bank 1 of the XYZ™ microcontroller:

Register & Signals	Function	Setting example for a 16-bits bus on Bank 1 (see User's Manual for timing details)
IO01BW[1:0]	I/O bus width	IO01BW[1:0]="1,0"
XA[23:0]	External address bus	User defined
XD[15:0]	External data bus	User defined
XIOCS_N[1]	I/O Bank 1 chip select	"0": Bank 1 selected
XOE_N	Output enable pin	
XWE_N	Write enable pin	
XBS_N[1:0]	External bus byte select: XBS_N[1] is for MSB, XBS_N[0] is for LSB	XBS_N[1] = "0" to select MSB XBS_N[0] = "0" to select LSB
IO01TYPE[2:0]	Access timing for I/O Bank 0 and 1	Set to "0,0,1" for: <ul style="list-style-type: none"> <li>Address Setup Time = 1 clock cycle,</li> <li>OE/WE Pulse Width = 4 clock cycles,</li> <li>Read off time = 3 clock cycles</li> </ul> (XYZ124001)
XWR pin	Data exchange direction	XWR = "0" : read from the external I/O device to the microcontroller, XWR = "1" : write from the microcontroller to the external I/O device.

**1.3- Memory mapping for external I/O access – Power savings estimate:**

The power savings allowed by the use of memory mapping is greatly dependent upon the user’s application profile (i.e. the ratio of I/O accesses versus internal computation).

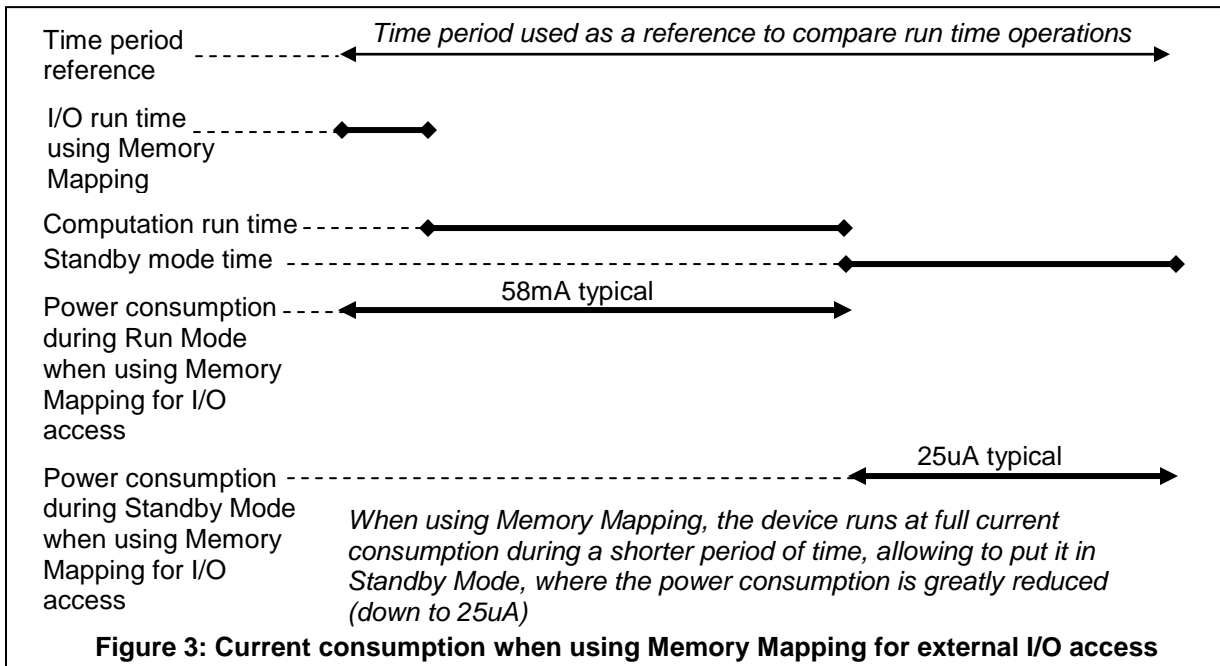
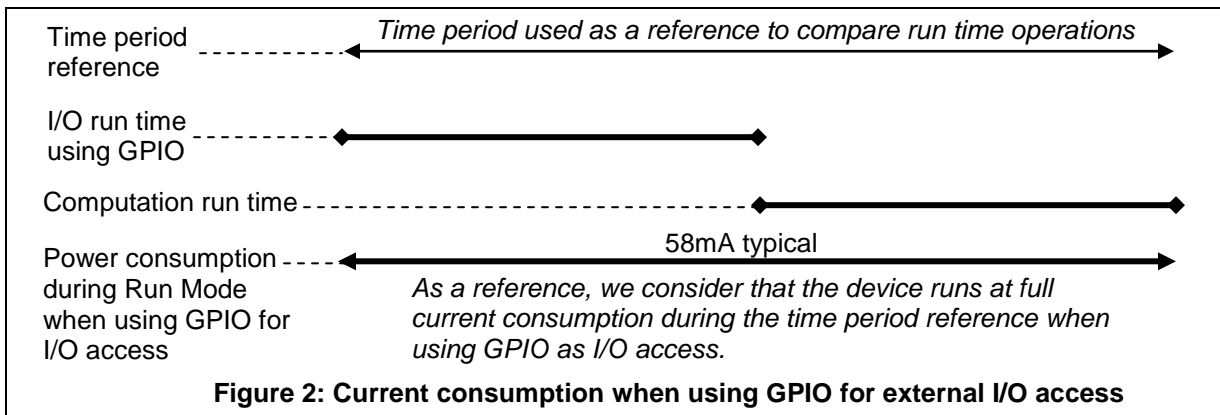
In order to estimate it, we consider that, for the amount of time saved by a faster I/O access, the device is put in Standby mode.

In figures 2 and 3 below, two consecutive tasks are executed:

- access to external device (shown as “I/O run time”)
- internal computation (shown as “Computation run time”)

In figure 2, we consider that those two consecutive tasks are executed in a certain period of time, called “time period reference”.

The device is here operating in Run mode (i.e. maximum power consumption) during the entire time period reference, while in figure 3, the device operates in Run mode for a shorter period of time (due to the shorter I/O access time), and is then put into Standby mode (i.e. low power consumption) for the remaining of the time period reference.



Using this estimation method, we can draw the following table that shows an estimate of the current consumption savings, at the device level, that the memory mapping method

can provide, depending upon the ratio of tasks between external data access versus internal computation:

Tasks Ratio		Run time (in fractions of Time Period Reference)						Power Savings Estimate (at the device level)	
I/O tasks	Computation tasks	Using GPIO			Using Memory Mapping			mA per second	%
		I/O	Computation	Standby	I/O	Computation	Standby		
60%	40%	0.6	0.4	0	0.07	0.4	0.53	30.6	53%
50%	50%	0.5	0.5	0	0.06	0.5	0.44	25.5	44%
35%	65%	0.35	0.65	0	0.04	0.65	0.31	17.9	31%
25%	75%	0.25	0.75	0	0.03	0.75	0.22	12.8	22%
15%	85%	0.15	0.85	0	0.02	0.85	0.13	7.7	13%
10%	90%	0.1	0.9	0	0.01	0.9	0.09	5.1	9%
5%	95%	0.05	0.95	0	0.01	0.95	0.04	2.6	4%

Note: XYZ12300x, 33MHz, 30pF I/O load

#### 1.4- Beware of zombies:

When you remove Vcc from sections of your design, some parts may try to come back to life on their own.

Many devices have now a protection diode that internally connects input pins to Vcc. This diode prevents input voltages from getting more than a few tenths of a volt above Vcc.

When in Standby mode, the XYZ™ microcontroller is still powered up, and if your design is such that a particular device, connected to the microcontroller's GPIO, is left with a floating Vcc, then this particular device may be powered up through this protection diode (see Figure3 below).

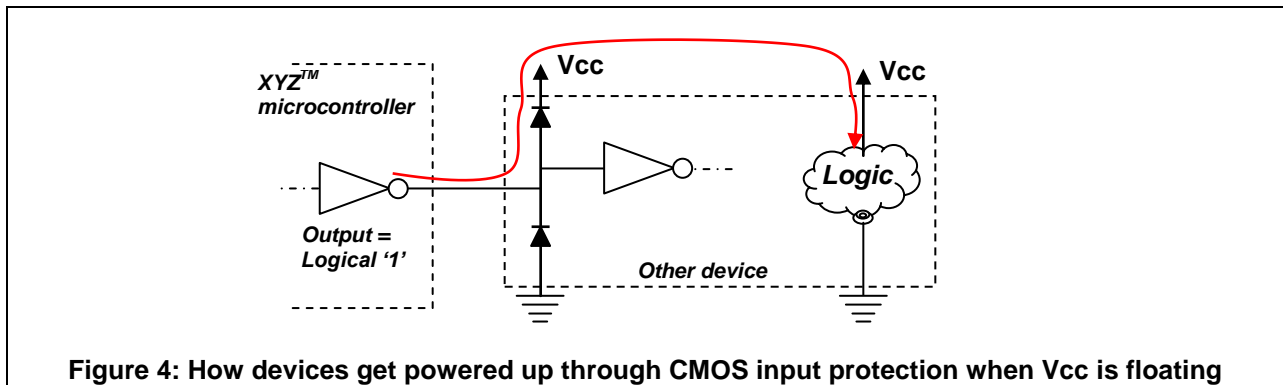


Figure 4: How devices get powered up through CMOS input protection when Vcc is floating

To prevent this, when setting your whole system in sleep mode, configure the XYZ™ microcontrollers GPIO as inputs before setting it into Standby Mode, using the Port Mode Registers (GPPMA,GPPMB, GPPMC, GPPMD and GPPME) as shown in the following table:

Register	Address	Bits	Setting
GPPMA	0xB7A01008	[7:0]	Set at '0' for Input
GPPMB	0xB7A01028	[7:0]	
GPPMC	0xB7A01048	[7:0]	
GPPMD	0xB7A01068	[7:0]	
GPPME	0xB7A01088	[9:0]	

#### 1.4- Some interfaces are not your friends:

Optically isolated I/O and electromechanical relays require quite a lot of current, and won't help in reducing power consumption.

Same thing for back-lighting and serial communications: they are not your friends. The user should limit the I/O and isolation to what is truly needed, and use pulsing techniques to reduce the current consumption of LEDs and displays.

The power savings achieved by pulsing LED depend upon their characteristics. For DC current supply, their relative luminous intensity versus DC forward current follow a linear curve, while for pulsed current supply, the relative efficiency of LEDs is dependent upon the peak forward current applied.

This is the reason why, for best pulsed operation and overall light output performance, LED vendors recommend a rectangular current waveform with a refresh rate equal to or greater than 100 Hz.

Although the designer will need to do some experiment to identify the best compromise of luminous intensity versus power supply savings for the target application, it is commonly admitted that such techniques allow reducing the power consumption of an LED by 25% (thus 1.25mA savings for a 5mA LED)

The XYZ™ microcontroller offers two ways of controlling the brightness (thus the current consumption) of your LED and displays: Pulse Width Modulation (PWM) and Timers.

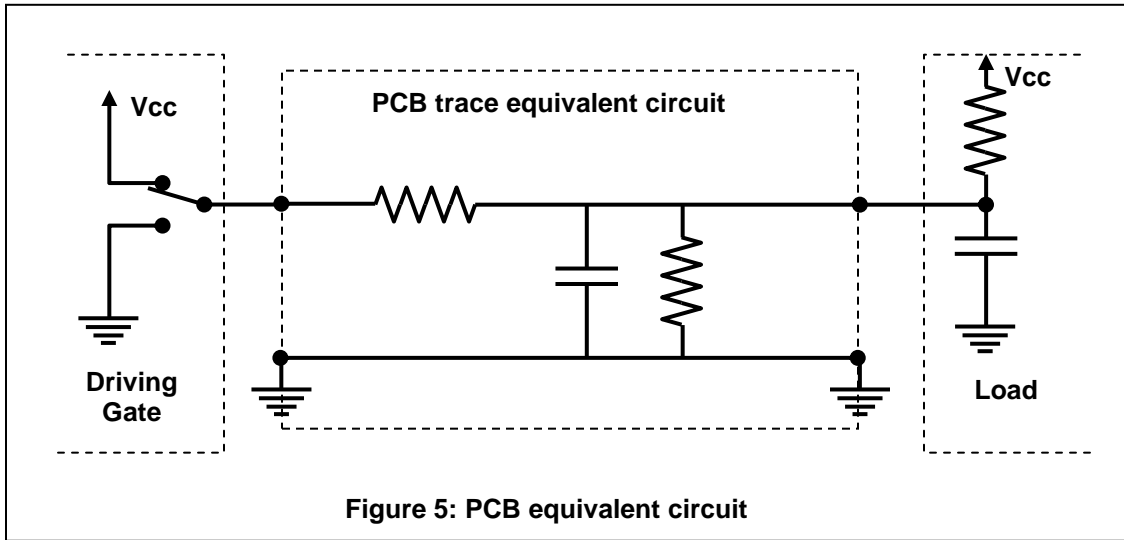
- Pulse Width Modulation (PWM): PWM is simple to use as it generates directly the pulsed signal aimed at driving the display or the LED.  
The sample code of Annex1 shows an example of PWM usage (in this case, to generate variable shaped pulse widths).
- Timers: when no PWM channel is available, the user can still use the XYZ™ microcontroller timers to schedule On/Off periods for LED and Displays.  
The use of timers to generate pulse width modulation signals may require a little more computation resources, and may not provide as much power savings as the use of PWM, depending upon your application.  
The sample code of Annex2 shows an example of timer based function (here a seven segment display counter).

**1.5- Power dissipation is also proportional to capacitance:**

**1.5.1- Shortening PCB trace length:**

While XYZ puts great emphasis in reducing the internal capacitance of the Advantage™ microcontrollers, this internal capacitance is beyond your control.

However, the overall power dissipation is also linked to the external load, which is a function of the PCB traces loading and other IC loads on the board (figure 5).



Each time the driving gate is switched from ground to  $V_{cc}$ , the PCB trace and load capacitors must be charged. When the gate switches from  $V_{cc}$  back to ground, the capacitors must be discharged. Since this energy does not get reused (it is dissipated as heat in the driving gate), every change in state causes a little loss in energy that needs to be compensated by over sizing the driving gate current.

This loss can be expressed as  $P = fV_{cc}C_L$ , where  $C_L$  is the load capacitance,  $P$  is the power loss, and  $f$  is the frequency.

As the PCB traces are getting longer, their capacitance and resistance is increasing, and so is the power loss.

A simple web-based estimation tool<sup>1</sup> allows estimating the power loss induced by different PCB trace lengths:

Trace length (inches)	Power loss (watts)
0.5	0.004
1	0.008
2	0.016
5	0.039
8	0.063

Note: Ambient temperature: 25°C  
Trace thickness: 2 oz. per sq. ft.

To reduce the power consumption, shorten the trace lengths of your PCB in order to make the external capacitance and resistance as small as possible.

<sup>1</sup> <http://www.geocities.com/CapeCanaveral/Lab/9643/TraceWidth.htm>

**1.5.2- Partitioning external bussing:**

As the  $P = fV_{cc}C_L$  equation applies to every signal on a bus, wide external busses should be avoided in order to reduce power consumption.

The load capacitance can vary between pins, but they are in general in the range of 4pF for an input and 12pF for a bi-directional pin.

The table<sup>2</sup> below illustrates the different levels of power consumption of a microcontroller's bus connected to a 1Mbit Flash device, and operating with a 4MHz bus frequency, based on the bus size:

	16-bits bus		8-bits bus	
	Bus cap. (uF)	Power (mW)	Bus cap. (uF)	Power (mW)
<b>Address pins</b>	238	4.8	252	5
<b>Data pins</b>	320	6.4	160	3.2
	<b>Total</b>	<b>11.2</b>		<b>8.2</b>

As shown in the above table, the estimated power savings of an 8-bits bus over a 16-bits bus is in the neighborhood of 26% in this case.

The table below shows an example of configuration for a 8-bits bus width external peripheral connected to Bank 1 of the XYZ™ microcontroller:

Register & Signals	Function	Setting example for a 8-bits bus on Bank 1 (see User's Manual for timing details)
IO01BW[1:0]	I/O bus width	IO01BW[1:0]="0,1"
XA[23:0]	External address bus	User defined
XD[15:0]	External data bus	User defined
XIOCS_N[1]	I/O Bank 1 chip select	"0": Bank 1 selected
XOE_N	Output enable pin	
XWE_N	Write enable pin	
XBS_N[1:0]	External bus byte select: XBS_N[1] is for MSB, XBS_N[0] is for LSB	XBS_N[1] = "0" to select MSB XBS_N[0] = "0" to select LSB
IO01TYPE[2:0]	Access timing for I/O Bank 0 and 1	Set to "0,0,1" for: <ul style="list-style-type: none"> <li>Address Setup Time = 1 clock cycle,</li> <li>OE/WE Pulse Width = 4 clock cycles,</li> <li>Read off time = 3 clock cycles</li> </ul> (XYZ124001)
XWR pin	Data exchange direction	XWR = "0" : read from the external I/O device to the microcontroller, XWR = "1" : write from the microcontroller to the external I/O device.

<sup>2</sup> Source: "Low-Power Design, by Mike Willey and Kris Stafford" – Embedded Systems Programming, December 28, 2001

## 2- Software Design tips

### 2.1- Lower the clock frequency for continuous running tasks

Lowering the clock frequency reduces power consumption for continuously running tasks, as shown in the following table:

Condition	Frequency	XYZ CPU Boards current consumption	
		XYZ123000	XYZ123003
Executing code	33 MHz	123 mA	227 mA
	16.5MHz	105 mA	178.7 mA
	8.25MHz	95 mA	167.9 mA
	4.125 MHz	89 mA	147 mA
	2.065 MHz	88.5 mA	141.3 mA

Note: Those measurements were made on XYZ XYZ123000 and XYZ123003 CPU-boards. The values reflect the power usage of the board as a system, and not the device by itself.

### 2.2- The clever use of Halt and Standby modes

The XYZ microcontrollers feature two power reduction modes:

- Halt mode: stops the clock signals to the following functional blocks: CPU, system bus, bus control circuitry, and memory controller interfaces to built-in RAM and external memory.
- Standby mode: stops the system clock oscillation entirely.

The Halt mode allows reducing the CPU board current consumption down to 149.7mA; when in Standby mode, this value is further reduced to 118.7mA.

While waking up from Halt mode is quasi-instantaneous (a few clock cycles), the wake-up cycle from Standby requires up to 24ms in order to allow the oscillator to stabilize before the code can start being processed again.

During these 24ms, the CPU board consumes the same current as in Idle mode (i.e. when no code is executed – 205.3mA<sup>3</sup>), but the code is not executed yet.

The two tables below detail the estimated current consumption of the XYZ123003 CPU board when using Standby and Halt modes in between task execution.

Current consumption estimate when using Standby Mode in between tasks						
Number of tasks	Average task duration (second)	Average Standby time (second)	Wake up time (second)	Task current consumption (mA)	Standby current consumption (mA)	Total current consumption (mA)
1	0.01	0.99	0.024	2.27	122.44	124.71
2	0.01	0.98	0.024	4.54	126.18	130.72
4	0.01	0.96	0.024	9.08	133.66	142.74
9	0.01	0.91	0.024	20.43	152.36	172.79
18	0.01	0.82	0.024	40.86	186.02	226.88
25	0.01	0.75	0.024	56.75	212.21	268.96
50	0.01	0.5	0.024	113.50	305.71	419.21
90	0.01	0.1	0.024	204.30	455.32	659.62

<sup>3</sup> Measured on the XYZ XYZ123003 CPU board

Current consumption estimate when using Halt mode in between tasks					
Number of tasks	Average task duration (second)	Average Halt time (second)	Task current consumption (mA)	Halt current consumption (mA)	Total current consumption (mA)
1	0.01	0.99	2.27	148.20	150.47
2	0.01	0.98	4.54	146.71	151.25
4	0.01	0.96	9.08	143.71	152.79
9	0.01	0.91	20.43	136.23	156.66
18	0.01	0.82	40.86	122.75	163.61
25	0.01	0.75	56.75	112.28	169.03
50	0.01	0.5	113.50	74.85	188.35
90	0.01	0.1	204.30	14.97	219.27

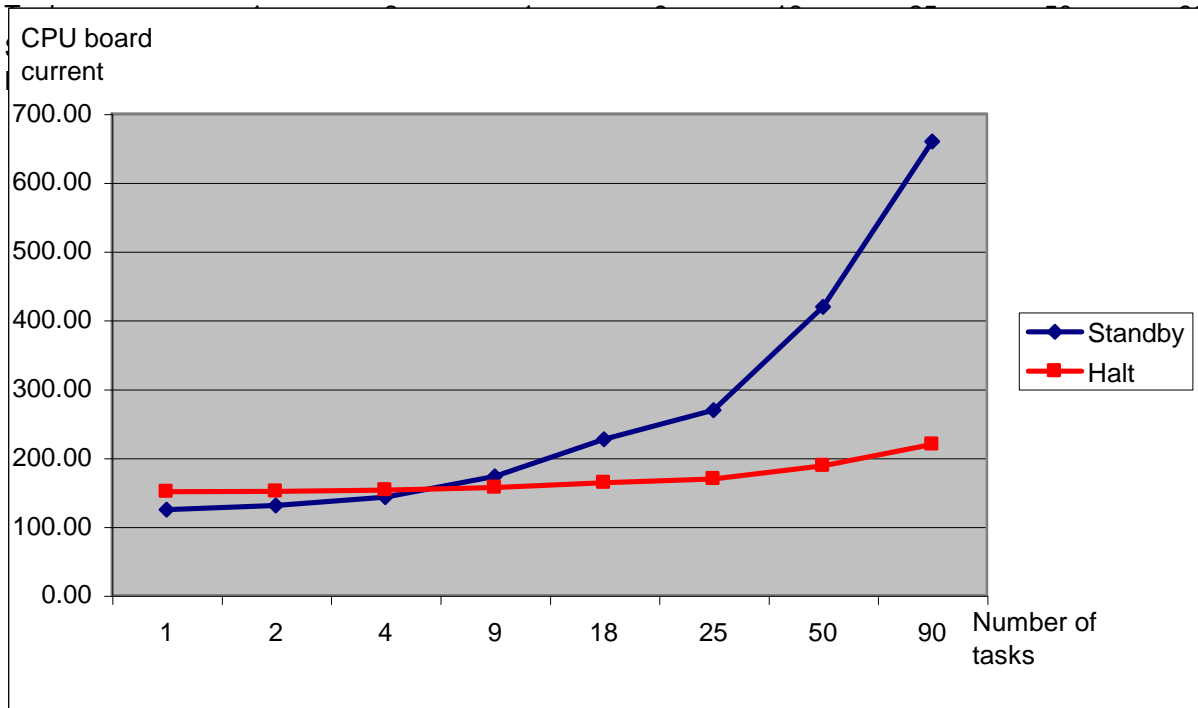


Figure6: CPU board Current consumption versus Number of tasks (10ms tasks) when using Standby and Halt modes in between tasks

The chart in figure 6 shows that, as the number of tasks increases for a defined period of time, the overall power consumption becomes higher when using Standby mode as opposed to Halt mode in between the tasks.

This phenomenon is due to the fact that waking up from Standby mode requires a certain period of time (24ms) for the oscillator to stabilize, period of time during which the code is not executed, but where the current consumption is almost as high (205.3mA<sup>4</sup>) as in Run mode (227mA).

To achieve the lowest power consumption for a given clock frequency, the user will need to carefully profile the code in order to identify which of Standby or Halt mode is more appropriate.

<sup>4</sup> Current consumption values are for the entire XYZ123003 CPU board

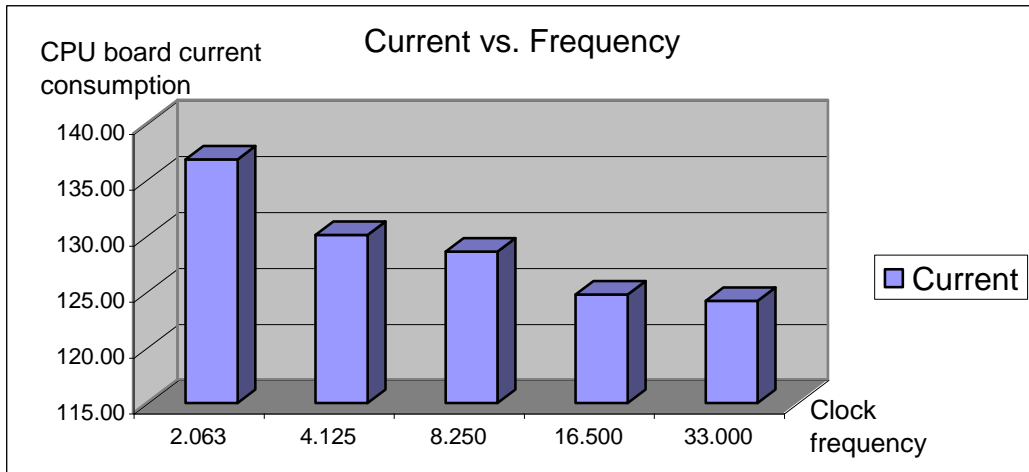
**2.3- Balance your options when idle times alternate with intermittent running tasks**

Lowering the clock frequency may not be your best option, depending upon the profile of your application.

The table below shows the current consumption for different application profiles (i.e. for different ratio of Running and Standby times) for the XYZ XYZ123003 CPU board.

For instance, a task that requires 0.05 second to be completed at 33MHz would require 0.8 second at 2.063MHz. Therefore, the respective Standby mode time goes from 0.95 second for a Run mode at 33MHz, to 0.2 second for a Run mode at 2.063MHz.

Current consumption estimate for the ML67Q4003 CPU board						
Frequency (MHz)	Profile (Task execution time in second)	Current consumption for this frequency (mA)		Current consumption for this profile (mA per second)		Total current consumption (mA per second)
		Run mode	Standby mode	Run mode	Standby mode	
2.063	0.8	141.30	118.70	113.04	23.74	136.78
4.125	0.4	147.00	118.70	58.80	71.22	130.02
8.250	0.2	167.90	118.70	33.58	94.96	128.54
16.500	0.1	178.70	118.70	17.87	106.83	124.70
33.000	0.05	227.00	118.70	11.35	112.77	124.12



**Figure7: CPU board current consumption versus clock frequency, using Standby mode in-between tasks (profile: one task per second)**

The figure 7 above shows that, for this specific case (one task to be executed per second), lowering the clock frequency would actually significantly increase the overall power consumption.

While one would believe that the lower power consumption would be achieved at 2MHz, the experiment shows that the best result is achieved with the higher clock frequency (33MHz in this case).

This phenomenon is explained by the fact that, when running at low clock frequencies, the device operates longer in Run mode (i.e. with high current consumption).

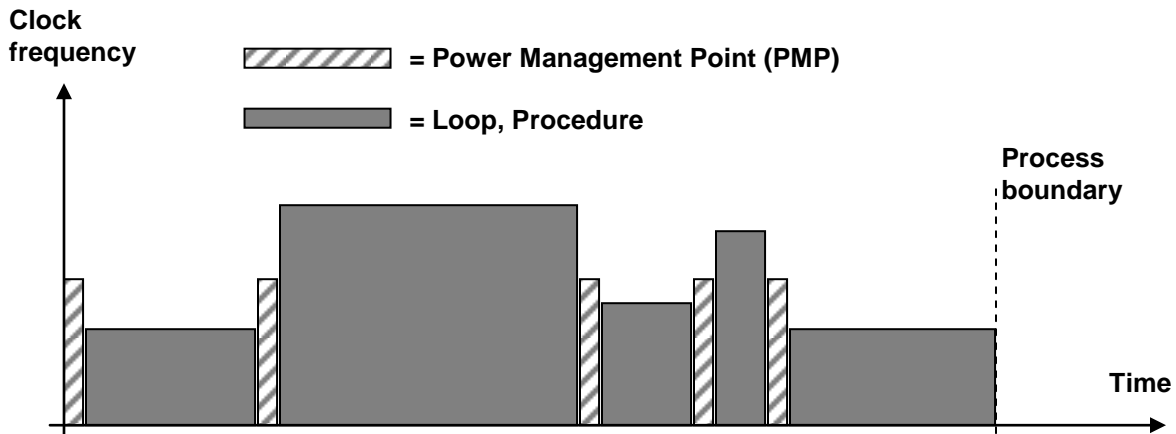
XYZ Company has partnered with third party software tools vendors to provide you with profilers that allow to test your code efficiency at different clock settings.

Task profiling when using RTOS:

Task profiling is specifically important when running a real time operating system (RTOS).

If power consumption is one of your major concern, make sure that your RTOS is power aware, and optimize the computation power with regards to each operation deadline. Run at a too high clock frequency, and you'll waste power by completing the task too early; run at a too slow clock frequency, and you'll miss the deadline.

When designing your own kernel, you should annotate your source code with temporal information (i.e. any observable action, like I/O access, also called "Power Management Points - PMP"): place this known temporal information at the beginning and the end of code sections (Loop boundaries, Procedure call sites, Alternative statements), as shown on the figure 8 below:



**Figure 8: Power Management Points (PMP) inserted in between loops and procedures running at different clock frequencies**

PMPs allow to:

- measure the progress of computation,
- measure the time consumed,
- recalibrate the remaining worst case execution time (WCET),
- adjust the processor speed for each loop or procedure.

XYZ Company has partnered with Real Time Operating System (RTOS) vendors to help you choose the right RTOS for your real time application.

As featured in the XYZ™ microcontroller, the Clock Gear Control Register (CGBCNT0) allows you to set, on the fly, the operating frequencies of the HCLK (CPU Bus) and CCLK (Peripherals).

Also use the CLKMD[1:0] pins on the XYZ124003 to set the operating frequency based upon the crystal used (see XYZ123001 Series/XYZ124001 Series User's Manual for detailed settings).

#### 2.4- Optimize the amount of code and data that you are handling at a time:

Accessing an external memory involves I/O buffers that require some power in order to be energized, and therefore would go against your goal of lowering the power consumption.

To reduce power consumption, you should place the Read/Write data in the internal memory of the XYZ™ microcontroller, instead of the external memory, using the Remap Control Register (RMPCON) register (see XYZ123001 Series/XYZ124001 Series User's Manual for detailed settings).

#### 2.5- Consider Cache lock for certain routines:

The XYZ™ microcontrollers allow you to lock in cache the routines that are repeatedly used.

In doing so, the internal CPU reads instructions or data in one clock cycle, and it reduces the number of clocks required to perform the operation.

However, this approach should be carefully experimented in your application, as, in some cases, it may reduce the overall throughput of the system, and increase its power consumption.

The code must be carefully profiled in order to identify the routines that can be cache-locked.

The XYZ XYZ12400x cache memory is split into four ways (or four storage locations); it is possible to lock the cache memory contents in one, two or three cache ways by setting LCK[1:0].

Further, by using the BNK[1:0] and F settings, it is possible to load instructions or data to the cache way to be locked.

See XYZ123001 Series/XYZ124001 Series User's Manual for detailed settings.

#### 2.6- Take full advantage of Thumb® mode:

The XYZ™ microcontrollers (XYZ123000, XYZ12300x, XYZ12400x) feature a 16-bits interface to external memory.

This feature is consistent with the use of the Thumb® mode, that allows compressing the code into 16-bits instructions without sacrificing computation performance.

By compiling in Thumb® mode, the instruction fetch is optimized, as the CPU reads 16-bits instructions out of a 16-bits memory.

Compared to ARM mode, the Thumb® mode reduces the number of clock cycles required to access code in the external memory, shortens the execution time, and allows the system to go back into sleep mode sooner (in turn reducing its power consumption).

XYZ Company conducted experiments showing that the performance increases when using 16-bits Thumb® mode as opposed to 32-bits ARM mode for the XYZ12300x (see table below).

Ok! Device	Operating Frequency	Program Space	16-bits Thumb mode Relative Performance (32-bit ARM mode reference performance: 100%)
ML67Q4003	33Mhz	Ext SRAM	121%
ML67Q4003	33Mhz	Int. Flash	129%

Note: Performance figures in this experiment are meaningful only with respect to the data within this experiment and not to be interpreted as absolute performance numbers to be compared to other architectures and/or systems.

These results are heavily case dependent, and the user must experiment different options in order to identify the more appropriate one for the system.

**2.7- Halt parts of the microcontroller whenever they are not needed:**

The XYZ™ microcontrollers allow to stop the clock to internal blocks when they are not needed.

The Block Clock Control Register (BCKCTL) allows stopping the clock to the A-to-D Converter, Pulse Width Modulation (PWM) generator, Timers, DRAM Controller<sup>5</sup>, DMA Controller, UART, Synchronous Serial IO (SSIO) and I2C.

The Clock Stop Register (CLKSTP) allows stopping the clock to the Serial IO (SIO).

XYZ Company has conducted experiment with regards to power savings induced by the use of Halt mode, as shown in the table below:

CPU Board current consumption and savings						
Frequency	ML674000			ML67Q4003		
	Run (mA)	Halt (mA)	Savings	Run (mA)	Halt (mA)	Savings
33MHz	123	100.7	18%	227	149.7	34%
2.063MHz	88.5	81.9	7%	147	125.3	15%

**2.8- Use Look-Up Tables (LUT) whenever possible:**

Look-Up Tables allow saving time over computing results; therefore, such functions aiming at providing a result can be executed in less number of clock cycles, and the system can go back in Standby mode sooner.

On another hand, and depending upon the application, Look-Up Tables tend to require more memory, so make sure that in saving clock cycles, you don't increase your memory size to a point that you need an additional memory chip, thus consuming more power.

The example source code shown in Annex3 is a standalone program running the popular HASH table lookup. The code shows how to add elements into the HASH table, look for a specific element in the table and see if it's a match or not.

<sup>5</sup> Remember to activate the DRAM Self Refresh operation by writing 110 in the DCMD Register.

## Annexes

## Annex1: Pulse Width Modulation sample code

```

/*****
/* PWM Sample Code */
*****/

#include "XYZ123001.h"
#include "common.h"
#include "irq.h"

/*
    PWM0 (Ds=10, De=100)

    duty(%)
    |           250       500
    +-----+-----+----->n(cycle)
    |
    | 1cycle=10ms
    | ->||<-
    De+ - - - - -
    |
    |
    Ds+ - - - - -
    |
    0 +-----+-----+----->time(ms)
    0       2500       5000
*/

/* constants */
#define MHz (1000000L)
#define CCLK (33*MHz) /* frequency of CCLK (Hz) */
#define PWM_CYC (100) /* frequency of PWM (Hz) */
#define PWCK (32) /* */
#define VALUE_OF_PWCY /* reload value of PWM */\
    (((65536*PWCK*PWM_CYC-CCLK)/(PWCK*PWM_CYC))
#define DUTY(n, Ds, De) /* duty(%) n,Ds,De:show above figure */ \
    (((long)(De)-(Ds))*(n)+(Ds)*((CYCLE)/2-1))/((CYCLE)/2-1)
#define VALUE_OF_PWR(duty) /* value of PWR duty:duty(%) */ \
    (((65536-(VALUE_OF_PWCY))*(duty)+(VALUE_OF_PWCY)*100)/100)
#define CYCLE 500

/* check VALUE_OF_PWCY */
#if (VALUE_OF_PWCY < 0 || 0x10000 <= VALUE_OF_PWCY)
#error Invalid value : VALUE_OF_PWCY
#endif

/* functions */
int main(void); /* main routine */
static void reg_irq_handler(void); /* registration of IRQ handler */
static void set_pwm(void); /* setup PWM */
static void pwm0_handler(void); /* PWM0 handler */

/* global variables */
UWORD value_of_pwr0[CYCLE];

```

```

/*****
/*  Entry point                                     */
/*  Function : main                               */
/*      Parameters                                 */
/*          Input  :  Nothing                      */
/*          Output :  0                            */
*****/
int main(void)
{
    /* initialize IRQ */
    init_irq();
    /* registration of IRQ handler */
    reg_irq_handler();
    /* setup TIMER */
    set_pwm();
    /* enable IRQ */
    irq_en();
    /* light LED */
    put_hvalue(GPPMB, 0x00ff); /* output mode */
    led_on(LED_START_PATTERN);
    /* setup port */
    set_hbit(GPCTL, 0x0020);
    /* start pwm */
    set_hbit(PWCON0, PWCON_PWR);
    /* infinite loop */
    for(;;)
        ;
    return 0;
}

/*****
/*  Registration of IRQ Handler                     */
/*  Function : reg_irq_handler                     */
/*      Parameters                                 */
/*          Input  :  Nothing                      */
/*          Output :  Nothing                     */
*****/
void reg_irq_handler(void)
{
    /* register IRQ handlers into IRQ handler table */
    IRQ_HANDLER_TABLE[INT_PWM0] = pwm0_handler;
    /* setup interrupt level */
    set_wbit(ILC1, ILC1_ILR12 & ILC1_INT_LV1); /* PWM0 (nIR[12]) -> level1 */
    return;
}

/*****
/*  setup of PWM                                    */
/*  Function : set_pwm                              */
/*      Parameters                                 */
/*          Input  :  Nothing                      */
/*          Output :  Nothing                     */
*****/
void set_pwm(void)
{
    int i;
    /* stop PWM */
    put_hvalue(PWCON0, 0); /* PWM0 */
}

```

```

/* clear PWM interrupt status register */
put_hvalue(PWINTSTS, PWINTSTS_INT0CLR|PWINTSTS_INT1CLR);
/* calculate duty value */
for(i=0; i<(CYCLE/2); i++){
    value_of_pwr0[i] = value_of_pwr0[(CYCLE)-1-i]
        = (UHWORDB)VALUE_OF_PWR(DUTY(i, 10, 100));
}

/* set duty value */
put_hvalue(PWR0, value_of_pwr0[0]);
/* setup cycle */
put_hvalue(PWC0, (UHWORDB)VALUE_OF_PWCY);
/* setup interrupt cause and clock */
put_hvalue(PWCON0, PWCON_CLK32|PWCON_INTIE|PWCON_PWCOV);
return;
}

/*****
/* PWM0 handler */
/* Function : pwm0_handler
/* Parameters
/* Input : Nothing
/* Output : Nothing
*****/
void pwm0_handler(void)
{
    static int index = 0; /* index of
value_of_pwr0[] */
    put_hvalue(PWINTSTS, PWINTSTS_INT0CLR); /* clear PWD0 interrupt */
    /* reset duty value */
    put_hvalue(PWR0, value_of_pwr0[index]);
    index++;
    if(index >= CYCLE)
        index = 0;

    return;
}
return;
}

```

**Annex2: Timer sample code**

```

/*****
/* Timer Sample Code */
*****/

#include "XYZ123001.h"
#include "common.h"
#include "irq.h"

/* constants */
#define MHz      (1000000L)
#define TMRCYC  (25)          /* interval of timer interrupt (ms) */
#define CCLK    (33*MHz)     /* CCLK (Hz) */
#define CLKGEAR (1)          /* clock gear */
#define CLKTMR  (CCLK/CLKGEAR) /* frequency of CLKTMR terminal (Hz) */
#define VALUE_OF_TMRLR /* reload value of timer */\
      ((0x10000L*(16*1000)-(TMRCYC*CLKTMR))/(16*1000))

#if ((VALUE_OF_TMRLR) < 0 || 0x10000 <= (VALUE_OF_TMRLR))
#error "Invalid value : VALUE_OF_TMRLR"
#endif

/* functions */
int main(void);          /* main routine */
static void reg_irq_handler(void); /* registration of IRQ handler */
static void set_timer(void);      /* setup of timer */
static void timer_handler(void);  /* timer handler */
void led_on(UHWORD);          /* light LED */

/* global variables */
static volatile int counter;
static volatile unsigned int led_count;

/* LED lighting pattern table */
UHWORD LED_TABLE [18] = {LED_0,LED_1,LED_2,LED_3,          /* "0","1","2","3" */
                        LED_4,LED_5,LED_6,LED_7,          /*
"4","5","6","7" */
                        LED_8,LED_9,LED_A,LED_b,          /*
"8","9","A","b" */
                        LED_C,LED_d,LED_E,LED_F,          /*
"C","d","E","F" */
                        LED_all,LED_off};                /* all on
,all off */

/*****
/* Entry point */
/* Function : main */
/* Parameters */
/* Input : Nothing */
/* Output : 0 */
*****/
int main(void)
{
    /* initialize IRQ */
    init_irq();

```

```

/* registration of IRQ handler */
reg_irq_handler();

/* setup of timer */
set_timer();

/* initialize LED */
init_led(); /* set output mode */

/* light LED start pattern */
led_on(LED_START_PATTERN);

/* initialize variable */
led_count = 0;
counter = 0;

/* enable IRQ */
irq_en();

/* timer start */
put_hvalue(TMEN, 0x01); /* enable timer (write '1' in TMEN[0]) */

while(1){
    if(counter >= 20){ /* timer interrupt occur? */

        counter = 0;

        /* light LED */
        led_on(LED_TABLE[led_count]);

        /* update led_count */
        led_count++;

        if(led_count >= 18)
            led_count = 0;

    }
}
return(0);
}
/*****
/* Registration of IRQ Handler */
/* Function : reg_irq_handler */
/* Parameters */
/* Input : Nothing */
/* Output : Nothing */
/* Note : Initialize of IRQ needs to be performed before this process. */
*****/
void reg_irq_handler(void)
{
    /* register IRQ handlers into handler table */
    IRQ_HANDLER_TABLE[INT_SYSTEM_TIMER] = timer_handler;

    /* setup interrupt level */
    set_wbit(ILC0, ILC0_ILR0 & ILC0_INT_LV1); /* system timer(nIRQ[0]) ->
levell */

    return;
}

```

```

/*****
/*  Setup of timer                                     */
/*  Function : set_timer                               */
/*  Parameters                                         */
/*      Input      :  Nothing                          */
/*      Output     :  Nothing                          */
*****/
void set_timer(void)
{
    put_hvalue(TMEN, 0x0); /* disable timer (write '0' in TMEN[0])*/
    put_hvalue(TMOVF, 0x01); /* clear overflow register (write '1' in
TMOVF[0])*/

    put_hvalue(TMRLR, VALUE_OF_TMRLR); /* set TMRLR */

    return;
}
/*****
/*  System Timer handler                               */
/*  Function : timer_handler                           */
/*  Parameters                                         */
/*      Input      :  Nothing                          */
/*      Output     :  Nothing                          */
*****/
void timer_handler(void)
{
    counter++;

    put_hvalue(TMOVF, 0x01); /* clear TMOVF register (write '1' in
TMOVF[0]) */
    return;
}

```

**Annex3: Look-Up-Table example**

```

/*****
/* Get a hash value for the specific string. */
/* Parameters: Input : char *string */
/* Output : hash value */
/*****
int HashIt( char *s )
{
    int hashval;
    for ( hashval = 0; *s != '\0'; )
    {
        hashval += *s++;
    }
    return ( hashval % HASHSIZE );
}
/*****
/* Lookup() gets a char string, converts it to the hashval, compares it to */
/* the hash table, and returns the pointer to the structure */
/* Parameters: Input : char *string */
/* Output : pointer to the struct of the table element */
/*****
struct nlist *Lookup ( char *s )
{
    struct nlist *np;
    int hashval;
    hashval = HashIt(s);
    for ( np = hashtab[hashval]; np != NULL; np = np->next )
    {
        if ( strcmp(s, np->name) == 0 )
        {
            return ( np );
        }
    }
    return ( NULL );
}
/*****
/* AddToTable() adds the element into the lookup table, searches the table */
/* first, converts it to a hash value, store add it to the hash table. */
/* Parameters: Input : Names and definitions */
/* Output : Nothing */
/*****
struct nlist *AddToTable( char *name, char *def )
{
    struct nlist *np;
    int hashval;
    if ( (np = Lookup(name)) == NULL )
    {
        np = (struct nlist *) alloc( sizeof(*np) );
        if ( np == NULL )
        {
            return ( NULL );
        }
        if ((np->name = strsave( name )) == NULL ) /*add name to table*/

        {
            return ( NULL );
        }
    }
}

```

```
    }
    hashval = HashIt( np->name );
    np->next = hashtab[hashval];
    hashtab[hashval] = np;
  }
  else
  {
      /* already there */
    free( np->def );
  }
  if ( (np->def = strsave(def)) == NULL )          /* save definition */
  {
    return ( NULL );
  }
  return ( np );
}
/*****
/* Main routine for test (allocate memory in internal SRAM)          */
int main( void )
/*****
{
  char string1[] = "Hello", string2[] = "World";
  char def1[] = "HelloDef", def2[] = "WorldDef";
  struct nlist *getName, *getNameAgain;
  AddToTable( string1, def1 );
  AddToTable( string2, def2 );
  getName = Lookup( string1 );
  if ( strcmp( getName->name, "Hello" ) == 0 )
  {
    printf("Hello!!! It's a match!\n");
  }
  else
  {
    printf("Hello!!! It's not a match!\n");
  }
  getNameAgain = Lookup( "Hallo" );
  if ( getNameAgain->name == NULL )
  {
    printf("Hallo!!! It's not a match!\n");
  }
  else if ( strcmp( getNameAgain->name, "Hallo" ) == 0 )
  {
    printf("Hallo!!! It's a match!\n");
  }
  else
  {
    printf("Hallo!!! It's not a match!\n");
  }
}
```

**References:**

“Low-Power Design, by Mike Willey and Kris Stafford” – Embedded Systems Programming, December 28, 2001

“High speeds and fine precision knock PCB traces off pedestal”, Rick Nelson, Test & Measurement World, January 2000

“Toward the Placement of Power Management Points in Real Time Applications”, N. AbouGhazaleh, D. Mossé, B. Childers, and R. Melhem, University of Pittsburgh, Department of Computer Science, 2001

“Dealing with tables”, C4Engineering (<http://www.bigprof.com/c4engineering/>).

XYZ Company – XYZ123000/XYZ12300x/XYZ12400x User’s Manuals

The information contained herein can change without notice owing to product and/or technical improvements.

Please make sure before using the product that the information you are referring to is up-to-date.

The outline of action and examples of application circuits described herein have been chosen as an explanation of the standard action and performance of the product. When you actually plan to use the product, please ensure that the outside conditions are reflected in the actual circuit and assembly designs.

XYZ Company assumes no responsibility or liability whatsoever for any failure or unusual or unexpected operation resulting from misuse, neglect, improper installation, repair, alteration or accident, improper handling, or unusual physical or electrical stress including, but not limited to, exposure to parameters outside the specified maximum ratings or operation outside the specified operating range.

Neither indemnity against nor license of a third party's industrial and intellectual property right, etc. is granted by us in connection with the use of product and/or the information and drawings contained herein. No responsibility is assumed by us for any infringement of a third party's right which may result from the use thereof.

When designing your product, please use our product below the specified maximum ratings and within the specified operating ranges, including but not limited to operating voltage, power dissipation, and operating temperature.

The products listed in this document are intended for use in general electronics equipment for commercial applications (e.g., office automation, communication equipment, measurement equipment, consumer electronics, etc.). These products are not authorized for use in any system or application that requires special or enhanced quality and reliability characteristics nor in any system or application where the failure of such system or application may result in the loss or damage of property or death or injury to humans. Such applications include, but are not limited to: traffic control, automotive, safety, aerospace, nuclear power control, and medical, including life support and maintenance.

Certain parts in this document may need governmental approval before they can be exported to certain countries.

The purchaser assumes the responsibility of determining the legality of export of these parts and will take appropriate and necessary steps, at their own expense, for export to another country.

Microsoft and Windows are registered trademarks of Microsoft Corporation. All other trademarked or copyrighted names that may be used are held by their respective owners.

Copyright 2003 XYZ Company

XYZ Company reserves the right to make changes in specifications at anytime and without notice. This information furnished by XYZ Company in this publication is believed to be accurate and reliable. However, no responsibility is assumed by XYZ Company for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of XYZ Company.

XYZ Web Site:  
<http://www.XYZsemi.com/us>

XYZ Stock No:

# **XYZ Company**

**Corporate Headquarters**

**555 South Helen Avenue  
Anytown, CA 94538 USA**

Tel: 510/555-5555

Fax: 510/555-5556